Increasing Resiliency through Priority Scheduling of Asynchronous Data Replication

Kevin P. Adams
NSWC
Dahlgren Division
AdamsKP@nswc.navy.mil

Denis Gračanin
Department of Computer Science
Virginia Tech
gracanin@vt.edu

Michael G. Hinchey
NASA Software Engineering Lab
Goddard Space Flight Center
Michael.G.Hinchey@nasa.gov

Abstract

Distributed systems commonly replicate data to enhance system dependability. In such systems, a logical update on a data item results in a physical update on a number of copies. The synchronization and communication required to keep the copies of replicated data consistent introduces a delay when operations are performed. In time-constrained systems or systems distributed over a bandwidth-constrained area, such operational delays generally prove unacceptable. Asynchronous data replication is commonly used to mitigate these delays. We look to develop a general solution for the introduction of an adaptive data replication scheduler to optimize asynchronous replications based on a user-developed priority model in overloaded situations. The solution uses a Multi-Layer Perceptron neural network to mimic the behavior of a historically optimal scheduler through functional approximation with its evaluation through simulation.

1. INTRODUCTION

A system replicating data asynchronously will replicate on a first-come first-served basis (FCFS) relative to the replication request. This is common in commercial asynchronous replication products such as those used in data vaulting where replications are scheduled to occur at predetermined times. The time period between the invocations of a replication on the same replication object is considered the replication interval. The term "replication object" is used to define a set of files that should be replicated together, thus having the same recovery point objective (RPO). This solution works fine as long as the system operates in an underloaded or fully-loaded condition.

In an over-loaded condition, a replication cannot complete during its replication interval. Current solutions allow (1) the current replication to be aborted; (2) the current replication to complete with the next replication of the object queued; and (3) the current replication is allowed to complete, skipping the next replication of the object. If the current replication is aborted, the updates propagated in the new replication must be recalculated from the last successful update. This approach does prevent queuing of replications, but has the potential of additional data loss in the event of a disaster. Allowing the current replication to complete provides the most recent RPO in the event of a needed recovery. After completion, if the new request is skipped, additional data loss could occur upon a catastrophic event. Conversely, if the new request is queued, delays are introduced into the system with the potential of increased future delays and data loss.

Recovery and impact from a catastrophic event can be greatly influenced by how current specific data is within a system. We propose that priorities can be assigned *a priori* to the data and these priorities can be used to create a benefit ratio allowing an optimized schedule of the replications for the available bandwidth during a replication interval. In order for the scheduling not to affect the RPO of an object, and the continuity of operations planning based on these RPOs, scheduling decisions need to be made in near real-time.

The remainder of this paper is structured as follows. Section 2 presents the issues that must be resolved and the proposed solution. Section 3 describes the simulation used to evaluate the solution including results from four simulations. Section 4 addresses scheduling adaptability. Section 5 concludes the paper.

2. THE APPROACH

The research problem can be described as follows: given a time interval, a bandwidth during this interval, and a set of objects for replication during the interval, each with a priority; determine the feasible set with the most benefit in real-time. The problem is exacerbated by the distributed nature assumed in the replication requests. The architecture assumes replications will be required from one or more sources with one or more communication channels possibly shared.

The major contribution of this work lies in providing a new methodology for passive asynchronous replication

that increases the resiliency of higher-valued data based on an admission control policy in a network that handles several classes of replication requests with different resource requirements over a shared network. Global admission control decisions are made in real-time with local state information, requiring minimal processing and communication traffic impacts on the operational system. The scheduler is intended to work with existing replication approaches.

2.1. Framework

In fault-recovery systems characterized by bandwidth constraints, acceptance of only minimal operational delays, or both, employing a passive asynchronous replication solution following the primary-secondary replication scheme in the primary-backup model is common. In this scheme, the passive replication is an asynchronous refresh technology where the model distinguishes one replica as the primary server, which handles all client requests. A write operation on the primary server invokes the transmission of an update message to the backup servers, updating the secondary replica(s). In other words, there will be only one source for each replication object and this source controls the updates to the replica(s) in a one-way write to a read-only framework. This is the framework assumed in this research. The backup replicas are read-only to avoid conflicts. In order to decrease the impact of the replication on the operational system and ensure there are no temporal issues, often referred to as temporal variance between data items, the updates are determined by point-in-time dependant replication where each replication object will have a defined RPO which is considered acceptable from a data-loss perspective.

There are two major shortcomings to the primary-secondary scheme from a resiliency perspective. The first is the assumption that all data objects have the same inherent value. This is rarely, if ever, the case as the first task in continuity of operations planning is to prioritize the objects to be recovered. How current specific data objects within a system are can greatly influence the recovery and impact from a catastrophic event. The second shortcoming deals with not meeting the RPO (a system requirement) on-time during periods of over-loaded conditions. When we define the RPO for each object and plan for each object's recovery in the continuity of operations plan, we rely on meeting the defined RPO for the recovery. The worst-case loss of data can be limited to two replication intervals for a replication object as long as the system operates in an under-loaded or fully-loaded condition. This two-replication interval limit can be readily seen under the assumption that all previous RPOs for that object have been met. The proof is by induction.

2.2. Replication Model

The proposed data replication model is comprised of three main components: replication request, scheduling of replications and the actual replication as shown in Figure 1. Processing elements make requests for replications on a known periodic. Each request will be for the replication of a set of replication objects. The replication service schedules and controls the replication process. The replication protocol performs the replication between the primary and the replica.

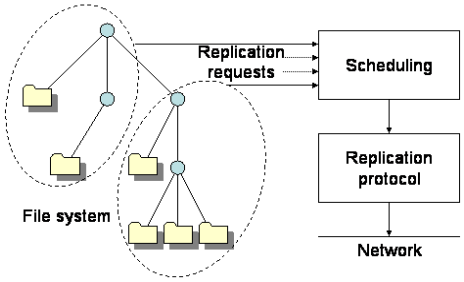

Figure 1: Data Replication Model

The granularity of the scheduled item is defined by the replication object. Replication objects are created based on the services provided within the system. They are user-defined containing one or more files, an assigned priority and a timing constraint. The priority can be viewed as a Quality of Service (QoS) parameter providing a relative assessment of how important it is to have the data replicated during a given interval, the timing constraint.

2.3. Priority Based Asynchronous Data Replication Scheduling

The introduction of a scheduler, or more properly an acceptance policy, into the process of asynchronous data replication allows the value of the data to be represented in the replication scheme through priorities. In over-loaded conditions, the goal is to maximize the accrued benefit of replications with end-to-end timeliness requirements.

The scheduler is looking to maximize the use of the limiting component, the bandwidth of the communications link in an over-loaded state. The admittance policy of the scheduler can be viewed as an optimization problem with replication object size and priority as parameters of the replication, while the link capacity and the length of time to complete the replication as the limiting factors. The optimization is to maximize the profit or accrued benefit where the benefit is defined to be the priority of the replication object. The trade-off in optimization of many lower-priority smaller items being selected over a large higher-priority item has the affect of creating a benefit ratio of benefit per unit of

bandwidth. An under-loaded condition is calculated by summing the size of the entire outstanding replication request. If the total is less than the available bandwidth during an interval, the system is under-loaded and all replications are admitted.

The proposed solution is to divide the scheduler into two parts, an off-line optimization to create the acceptance policy and a real-time implementation of the acceptance policy [1]. By "off-line" we mean in peripheral processing that is not part of the operational system. The off-line optimization of replication request can be modeled as variants of a 0-1 Knapsack Problem (KP) given the number of sources, destinations and paths from source to destination. An optimal scheduler requires knowledge of all replication requests for optimal scheduling; thus scheduling must occur at intervals. We define these intervals as the length of time allocated for the replication. Thus the items in the queue to be scheduled are the items for which a replication request has been made, the replication has not been initiated and the replication interval has not expired. Solving the KP provides the optimal feasible list of objects to be replicated in a given interval. The solutions to these integer-programming problems are used to train neural networks. Once trained, the weights from the training network are used in a MLP network to make real-time binary decisions on acceptance of replication requests, mimicking the behavior of the historically optimal scheduler given a fixed set of input parameters. The same input parameters are used in training the network. In order for the scheduler to continue to perform well as the replication patterns change the network must be retrained. As the in-line systems operates, the off-line systems continues to calculate the optimal solution for previous data measuring the performance of the in-line component, continuing to train the MLP and updating the in-line MLP as necessary. If there is no real-time requirement for the scheduling, the off-line component can be placed in-line and the MLP network is unnecessary. We consider timeliness requirements that generalize the hard real-time timing constraint to encompass non-binary timing constraints. The completion of a replication at any time yields a benefit specified by a benefit function, even if the replication is late.

The evaluation of the approach begins with a scheduler that provides an optimum schedule for replicating a single read-only replica over a variable bandwidth connection. A 0-1 KP models this behavior. The model is used to generate historically optimal data for training the MLP neural network used to implement the acceptance policy. The 0-1 KP is described in Section 2.4. Expanding the model to support multiple paths from the source to the destination can be modeled as a Multiple Knapsack Problem (MKP). The MKP is described in Section 2.5. In modeling multiple sources, multiple destinations or multiple paths, multiple instances of the KP or MKP are used. Table 1 summarizes these scenarios. Simulations based on the four unique models, 0-1 KP, MKP, multiple 0-1 KP and multiple MKP are presented in Section 3. KP is NP-hard and can be solved a number of ways. Keller, Pferschy and Pisinger [6] provide a further discussion of solving many variants of the KP.

TABLE 1: MODELS OF REPLICATION CASES

Source	*Destination*	*Paths*	*Bandwidth*	*Optimization Model*
Single	Single	Single	Variable	0-1 KP
Single	Single	Many	Variable	MKP
Single	Many	Single	Variable	Multiple 0-1 KP
Single	Many	Many	Variable	Multiple MKP
Many	Single	Single	Variable	Multiple 0-1 KP
Many	Single	Many	Variable	Multiple MKP
Many	Many	Single	Variable	Multiple 0-1 KP
Many	Many	Many	Variable	Multiple MKP

2.4. Zero-One Knapsack Problem Model

An instance of the 0-1 KP can be defined by the capacity c and a set of n items where an item i is described by its profit p_i and weight s_i. A subset of items is selected such that the total profit of the selected items is maximized and the total weight does not exceed c. The KP can be formulated as a solution for the following integer program:

$$\text{Maximize } O = \sum_{i=1}^{n} p_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} s_i x_i \le c \qquad (1)$$

$$\text{and } x_i \in \{0,1\}, i = 1,...,n$$

The profit p_i belongs to the set of priorities $\{P_1, \ldots, P_m\}$. Higher priority values represent higher priority items. The decision vector $\mathbf{x}$ identifies which items are to be inserted into the knapsack. A value of one identifies insertion. All of the coefficients are positive integers and O is the objective function for maximal benefit in the system. The weight of each item is less than the capacity so that it is possible to be scheduled. If an object's weight is greater than the current capacity, the object cannot be scheduled during the current replication interval. Finally, the summation of weight for all items submitted to the scheduler must be greater than the capacity. In the event that the weight of all items submitted to the scheduler is smaller or equal to the capacity, all items are scheduled.

2.5. Multiple Knapsack Problem Model

MKP is a generalization of the 0-1 KP from a single knapsack to m knapsacks with possibly different capacities. The objective is to assign each item to at most one of the knapsacks such that none of the capacity constraints are violated and the total profit of the items put into the knapsacks is maximized. Given a set of items $N=\{1,\ldots,n\}$ with profits p_j, and weights w_j, for $j=1,\ldots,n$ and a set of knapsacks $M=\{1,\ldots,m\}$ with positive capacities c_i, $i=1,\ldots,m$, the objective is to select a subset $N' \subseteq N$ such that the items of N' can be assigned to the knapsacks without exceeding the capacities and the total profit of N' is maximized. Therefore:

$$O = \max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$$

$$x_{ij} \in \{0,1\}, i = 1,...,m,\ j = 1,...,n$$

$$\text{Subject to } \sum_{j=1}^{n} w_j x_{ij} \le c_i, i = 1,...,m \qquad (2)$$

$$\sum_{i=1}^{m} x_{ij} \le 1,\ j = 1,...,n$$

where $x_{ij} = 1$ if item j is assigned to knapsack i and zero otherwise. The profit p_j belongs to the set of priorities $\{P_1,\ldots,P_k\}$ and $\mathbf{x}$ is the decision vector as described in the previous section. All of the coefficients are positive integers and O is the objective function for maximal benefit in the system. An item cannot be scheduled in a given replication interval if its weight is greater than the available capacity in each knapsack.

2.6. Multilayer Perceptron

MultiLayer Perceptrons (MLPs) are feed-forward neural networks trained with the standard back-propagation algorithm. They are supervised networks so they require a desired response to be trained. They learn how to transform input data into a desired response, so MLPs are widely used for pattern classification [2]. Hornik et al. [5] and Funahashi [4] showed in 1989 that single hidden layer networks are capable of approximating any continuous function to any given accuracy, provided that sufficiently many processing elements are available, thus MLPs can approximate virtually any input-output map.

A MLP is a network of simple neurons called perceptrons. The perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then possibly putting the output through some nonlinear activation function. A typical MLP network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-by-layer. Bishop provides a detailed discussion of neural networks [3].

3. SIMULATION

Simulation results for the four unique scenarios as outlined in Table 1 are presented. The simulations consist of three components: submission of replication request, scheduling the replications, and finally, the replication. The case studies are an evaluation of the scheduling portion of the simulation.

Our Monte Carlo simulation for the scheduling uses three inputs and calculates one output for each replication request. The input parameters are: *Priority*, *Size*, and *Capacity_used*. The output value, *A/R*, is our binary decision to replicate this interval or be queued. Replication requests have a timing requirement and a priority. The scheduler uses the priorities of the replication request as the scheduling criteria, maximizing the priority and also maximizing the use of the link capacity. Objects submitted for replication consist of: the full pathname for the data file(s), the priority of the object, and the size of the object. For the simulations, the priority of each object is 2, 3, or 4; the higher the number the higher the priority. A uniform distribution is used to generate random priorities for each replication object. To take into consideration the effects of file modifications requiring a variable size update over multiple replication intervals, the simulation uses a uniform random percentage of each file to be replicated when it is submitted to the scheduler. The *Capacity_used* parameter is the percentage of the total capacity used when the replication request is made for the object. The *Capacity_used* parameter for a replication object is the ratio of current capacity used over total capacity of the channel. It is the sum of

1. The bandwidth of the channel (a constant) in bytes minus the bandwidth allocated for replications this interval in bytes. The bandwidth allocated for replication this interval is a uniform random variable whose range varies by the scenario;
2. The cumulative bandwidth used for replications at this point in the current replication interval; and
3. The size in bytes of the current replication request.

The sum is divided by the capacity of the channel for the interval. The inputs from our simulation are optimized by the off-line processing, producing a comma-separated record for each replication request. Each record contains six fields: admittance or rejection (*A/R*), priority of request (*Priority*), size of request (*Size*), the percentage of the capacity used when the item is queued for transmission (*Capacity_used*), the full file pathname (*File*) and the destination (*Dest*).

These records are used in training the MLP network. The input parameters of the MLP are: *Priority*, *Size*, and *Capacity_used*. All of these parameters are numeric. The *A/R* output parameter is the decision vector and is used to train the desired output (Figure 2).

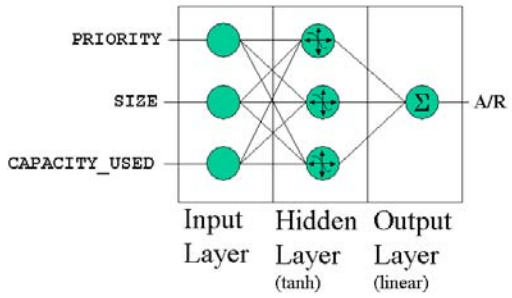

Figure 2 – Signal-flow graph of a single hidden layer MLP with three hidden elements

The MLP provides real outputs for the accept/reject criteria. To use the MLP results as the acceptance policy for the scheduler, the results are rounded to provide integer results using the standard rounding rules:

$$-0.5 \le x_i < 0.5,\ x_i=0$$
$$0.5 \le x_i < 1.5,\ x_i=1$$

otherwise $x_i=2$ (error condition).

Once the MLP network is trained, the weights for each of the network elements are saved. These weights are loaded into the in-line MLP for real-time decision-making. The in-line neural network is validated by using data not used in the training.

As a basis of the case studies, 528 Replication Objects were selected from a single system. The distribution of the replication objects is 174 at priority 4, 13 at priority 3 and 341 at priority 2. Each case study described in this section will simulate 100 replication intervals. During each interval, the selection of a random number of random replication objects (0-528) from the list of replication objects will be made. The size of the replication for each replication object is a random percentage of the object, 0-100% representing the percentage change in each object since the last replication. The off-line optimization results in a number of exemplars (samples) in each case that are randomized and used to train and verify a MLP neural network to recognize the scheduling pattern. The neural network used in each case study is a single hidden layer MLP with four hidden processing elements. The MLP was trained in each case using 80% of the sample data. The remaining 20% of the sample data is used for Out of Sample Data Validation. Each training session is for 1000 epochs. The value of Ã calculated in each case study is the percentage of decisions made the same as an optimal clairvoyant scheduler. After the MLP is validated, it is verified by a test using new data from each simulation for 25 replication intervals. The results of the four case study verifications are presented in Figure 3 (Ã of 99.46%, 99.21%, 99.47% and 99.69% respectively).

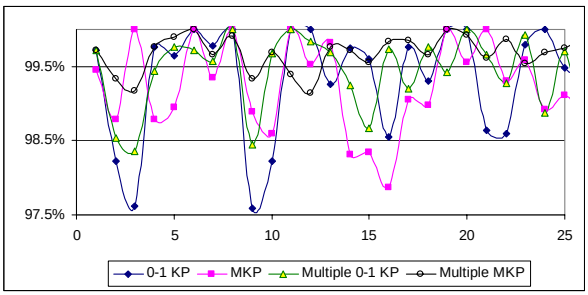

Figure 3 – Case Study Results

3.1. Case 1: 0-1 KP Model

The first case study is a single replication source to a single destination over a single variable bandwidth communications channel. The bandwidth ranges between 0 and 983,010 bytes per replication interval. The optimization based on the Pisinger's minimal algorithm for the 0-1 KP [8] took 8.366 seconds and results in 31,350 exemplars (samples); of which 25,080 are used for training and 6,270 for Out of Sample Data Validation. The optimal results for the validation data are 4,548 objects accepted and 1,722 objects rejected. The results of the simulation were: 6,238 correct decisions; 34 incorrect decisions; resulting in a validation Ã of 99.49%.

3.2. Case 2: MKP Model

The second case study is a single replication source to a single destination over three variable bandwidth communications channel. The bandwidth ranges between 0 and 327,680 bytes per interval per path. The optimization is based on the Pisinger's exact algorithm for large MKP [7] took 8.116 seconds and results in 32,101 exemplars; of which 25,681 are used for training and 6,420 for Out of Sample Data Validation. The optimizations determine which path to optimally replicate the data. The MLP implements the acceptance policy, once accepted, the real-time scheduler uses a first fit algorithm to assign an outgoing path. The evaluation of the MLP is for the acceptance policy verses a routing policy. The optimal results for the validation data are 4,373 objects accepted and 2,047 objects rejected. The results of the simulation were: 6,362 correct decisions; 58 incorrect decisions; resulting in an Ã of 99.1%.

3.3. Case 3: Multiple 0-1 KP

The third case study models multiple 0-1 KP simultaneously. The 0-1 KP model is used when a single

communications path is available. The multiple 0-1 KP model is used for single sources to multiple destinations over single paths; for multiple sources, each to a single destination over single paths; and for many sources to many destinations with each source to destination having a single path.

The results presented are for three replication sources to a destination, each with a single variable bandwidth communications channel. The bandwidth ranges between 0 and 983,010 bytes per replication interval per destination. The optimization based on Pisinger's minimal algorithm for the 0-1 KP [8] as in the first case study took 6 minutes 15.946 seconds for the three sources total and results in 93,584 exemplars (31,565; 31,001; 31,018); of which 74,867 are used for training and 18,717 for Out of Sample Data Validation. The weights from the trained MLP are uploaded into the MLP neural networks at each source. The results are presented in Table 2.

TABLE 2 – CASE STUDY 3 RESULTS

	Accept	*Reject*	*Correct*	*Incorrect*	$\tilde{A}$
A	4,478	1,841	6279	40	99.37
B	4,452	1,582	6,003	31	99.49
C	4,661	1,703	6,332	32	99.50
∑	13,591	5126	18,614	103	99.45

3.4. Case 4: Multiple MKP Model

The final case study models multiple MKP simultaneously. The MKP model is used when multiple communication channels are available. The Multiple MKP model is used for single sources to multiple destinations over multiple paths; for multiple sources, each to a single destination over multiple paths; and for many sources to many destinations with each source to destination having multiple paths.

The results presented are for three replication sources to one destination with each source having four variable bandwidth communications channels. Each source has two point-to-point paths and two channels shared between the three sources. The bandwidth ranges between 0 and 245,760 bytes per interval per path. The off-line optimization is based on Pisinger's exact algorithm for large MKP [7], as in the second case study. The off-line processing first optimizes the point-to-point connections as in the second case study. Rejected objects from all sources are then optimized for replication over the two shared channels. The off-line processing results in 115,402 exemplars (39,945; 37,618; 37,839); of which 92,322 are used for training and 23,080 for Out of Sample Data Validation. As in the second case study, the MLP implements the acceptance policy, once accepted, the real-time scheduler uses a first-fit algorithm to assign an outgoing path. The results are presented in Table 3.

TABLE 3 – CASE STUDY 4 RESULTS

	Accept	*Reject*	*Correct*	*Incorrect*	$\tilde{A}$
A	4,469	3,535	7,984	20	99.75
B	4,251	3,291	7,542	0	100
C	4,258	3,276	7,534	0	100
∑	12,978	10,102	23,060	20	99.91

4. SCHEDULING ADAPTATION

The results shown in Section 3 demonstrate how effective the functional approximation supplied by the MLP can be if the new request follows the pattern of the previous requests used in training. In Section 3, the validation data was a random sampling of original data set removed before training. This approach guarantees that the validation data will follow the same pattern as the training data. But of course, this will not always be the case. In this Section we present a reactive solution that can be used to update the neural network.

The off-line optimization results are used to train a neural network of the appropriate acceptance policy. The weights associated with the parameters of the network, calculated from the training process, are loaded into the MLP network used to make real-time binary decisions on acceptance of each replication request. In order to "boot strap" the initial training process, the initial set of weights for the MLP are initialized to pre-calculated values of a given replication pattern if it is known or random values if not. As replication requests are made to the real-time MLP, the input parameters and the MLP results are logged. At predefined replication intervals, the log is processed by the off-line component and the log restarted on the in-line component. The off-line component calculates the optimal solution and uses these results to continue supplementary training of the off-line MLP. Each iteration of training begins with the weights from the previous training session. The off-line processor also compares the optimal results with the decisions made by the in-line MLP.

The results of the off-line training results in a new set of weights taking into account previous data and the new set of training data. These weights can be loaded into the in-line MLP in near real-time by sending a signal to the in-line MLP to reload its weights. The process of updating the weights of the in-line MLP can occur at predefined intervals or when the results skew too much from optimal, an indication that a new replication pattern is emerging. In order to be able to train off-line and simply load the weights into the in-line MLP, the neural network designs must be identical.

The effectiveness of this approach has been initially evaluated under simulation. In the simulation, the custom MLP network used in the case studies was trained under a variable bandwidth in the range of 0 – 1048544 with same parameters as used in the case study presented in Section 3.1. The initial training weights were randomly assigned. Training consisted of 100 replication intervals for 100 epochs.

Figure 4 shows a simulation run of 1000 replication intervals with updates from the off-line training MLP every 100 replication intervals. Training is for 100 epochs. The spikes indicating performance drops are single replication intervals. For example, the three instances where performance drops below 88% are at intervals 300, 429 and 603. The reason for the spikes appears to be a small number of replication requests during these intervals so an incorrect decision carries a higher weight during these intervals. Figure 5 presents the same data as a cumulative percentage of the optimal solution. As the simulation continues over time the decision-making improves toward optimal with less variability as the in-line MLP receives updated weights from off-line training.

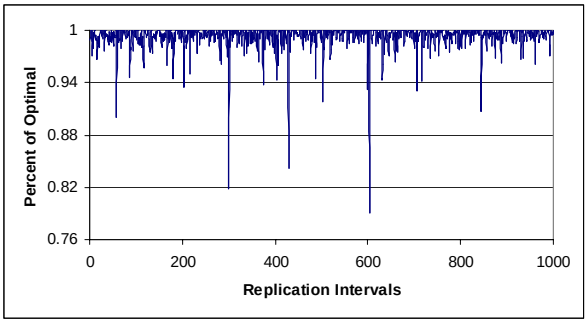

Figure 4 – Adaptive Scheduling Static Pattern

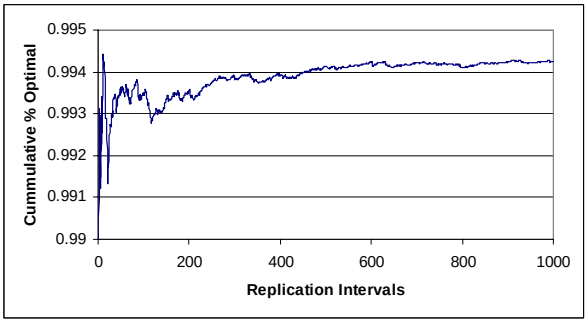

Figure 5 – Cumulative Percentage of Optimal

Figure 6 shows the effect of changes in the replication request pattern every 100 replication intervals. The range of the variable bandwidth was decreased each 100 replication intervals for a random amount between 25 - 75%. This changes the bias of the sample. A dramatically changing pattern, like the one during replication request intervals from 500 to 600 can greatly affect the performance of the system, but after retraining performance quickly improves.

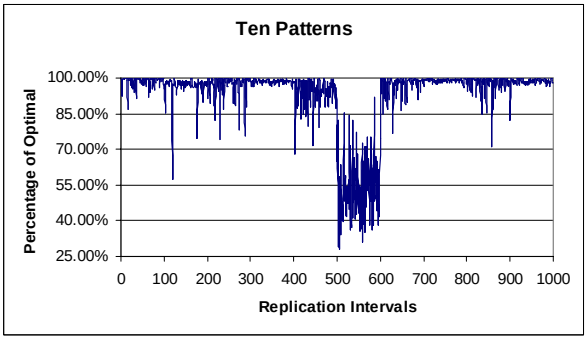

Figure 6 –Scheduling with a Dynamic Pattern

5 CONCLUSION

The simulations have demonstrated the approach of a real-time optimization using a Multi-Layer Perceptron Network to perform extremely well for determining acceptance given a QoS policy for a replication scheduler in real-time. The main issue with this approach is ensuring that the patterns in the data are adequately represented in the training data and recognizing when a pattern is changing or a new pattern is present. The presented solution continues the offline processing of the data, comparing scheduler and optimal results with subsequent updating of network weights as required reactively. When patterns are known and repeating, changing to a predetermined MLP works well.

REFERENCES

1. Adams, K., D. Gračanin, and D. Teodorović, "A Near Optimal Approach to Quality of Service Data Replication Scheduling", *Winter Simulation Conference (WSC04)*, Washington D.C., Dec. 2004, pp. 1847-1855.
2. Bishop, C. M., "Neural Networks for Pattern Recognition", Oxford University Press, New York, 1995.
3. Bishop, C. M., "Neural Networks and Machine Learning". Springer, 1998.
4. Funahashi, K., "On the approximate realization of continuous mappings by neural networks", *Neural Networks*, 2(3):183-192, 1989.
5. Hornik, K., M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, 2(5):359-366, 1989.
6. Keller, H., U. Pferschy, and D. Pisinger, "Knapsack Problems", Springer-Verlag, 2004.
7. Pisinger, D., "An exact algorithm for large multiple knapsack problems", *European Journal of Operational Research*, 114:528-541, 1999.
8. Pisinger. D., "A minimal algorithm for the 0-1 knapsack problem", *Operations Research*, 45:758-767, 1997.